Cornell University

Department of Physics

Phys 480/680, Astro 690 February 22, 2000

Computational Physics, Spring 2000

Homework Assignment # 4

(Due Thursday, March 2 at 8:40 am in Lab.)

Agenda and readings for the week of February 1:

All readings from *Numerical Recipes: The Art of Scientific Computing*, 2nd edition (in Fortran *or* C).

- Lec8, 02/29: Interpolation; Solution of many non-linear equations in many unknowns (Broyden approaches)
  Reading: 9.6–9.7
- Lab3, 03/02: Full atomic spectra for periodic table (one-electron atoms)

Goal: This homework develops a fully automated solver which finds all of the energy states for any specified external potential "V[]".

Contents

1 Finding the right bounds **2**
- 1.1 Modifying "schint" . . . 2
- 1.2 Modifying "func_Schrodinger" . . . 2
- 1.3 New routine: "func_SchrodingerNodes" . . . 2
- 1.4 Fixing "rtbisp480" . . . 3
- 1.5 Debugging . . . 3

2 "getEs" **3**
- 2.1 New routine . . . 3
- 2.2 Fixing "zriddrp480" . . . 4
- 2.3 Debugging . . . 4

3 "getallEs" **4**
- 3.1 New routine . . . 4
- 3.2 Debugging . . . 6

1 Finding the right bounds

In the previous homework, you developed a solver which rapidly finds a particular electron energies once given a lower and an upper bound. This problem renders into software the technique discussed in lecture for finding such bounds.

1.1 Modifying "schint"

```
int schint(double *Psi, double *Psip, double Psiout[],
           int k1, int k2, double Veff[], double E,
           double r[], double dr[], int N);
```

Return value:

- number of nodes in solution between r[k1] and r[k2]

Output:

- Psi, Psip: values of $\Psi(r[k2])$ and $\Psi'(r[k2])$
- Psiout[]: not used (yet!)

Input:

- Psi, Psip: values of $\Psi(r[k1])$ and $\Psi'(r[k1])$
- k1, k2: indices of initial and final integration points (r[k1] $\rightarrow$ r[k2])
- Veff[]: effective potential on grid
- E: proposed electron energy
- r[], dr[], N: standard grid information

Modify your "schint" routine to conform to the above prototype. The modified routine (now a function subroutine if you are working in **Fortran**) should now return an integer equal to the number of "nodes" (zero crossings of the solution Psi) in the range from k1 to k2. Note that the routine also accepts an array Psiout, which we shall use in a later problem set.

1.2 Modifying "func_Schrodinger"

Modify the outward, k=0 $\rightarrow$ k=N, version of "func_Schrodinger" to be able call your modified "schint" by placing a dummy argument in the slot for Psiout. (If you are working in **C**, just use NULL for the Psiout argument in the call. If you are working in **Fortran**, you can just send any double variable, such as Psi.) The prototype for "func_Schrodinger" should not change.

1.3 New routine: "func_SchrodingerNodes"

```
double func_SchrodingerNodes(double E,
                             int kmatch, double Veff[],
                             double r[], double dr[], int N)
```

Return value:

- (number of nodes at energy E) – (kmatch)

Input:

- E: proposed electron energy
- kmatch: targeted number of nodes in solution
- Veff[]: effective potential on grid

- r[], dr[], N: standard grid information

Write a routine "func_SchrodingerNodes", which is very much like "func_Schrodinger", *except* that it returns the number of nodes reported by schint *minus* the value of the argument kmatch. In **C**, for instance, your return statement will appear as something like:

```
return ((double) (schint(&Psi,&Psip,NULL,0,N,Veff,E,r,dr,N)-kmatch));
```

Note: Although the return value is always an integer, we return it as a double so that we can use "rtbisp480" routine without modification.

1.4 Fixing "rtbisp480"

The numerical recipes routine "rtbis" has a small bug: If one of your two initial guesses x1 or x2 for the solution gives f(x)=0, the routine prints an error message and halts the program. Modify "rtbisp480" to test for these conditions and to return the appropriate value, x1 or x2, as the solution.

1.5 Debugging

Write a main program which makes the call

```
E=rtbisp480(func_SchrodingerNodes,-0.9,0.,1e-12,n,Veff,r,dr,N);
```

with the potential Veff[k]=-1./r[k] for a total number of intervals N=200 and our standard change of variables. Verify that for n=0, 1 and 2 you get values in the ranges $(-0.9, -0.5]$, $(-0.5, -0.125]$ and $(-0.125, -0.055556]$, respectively.

2 "getEs"

2.1 New routine

```
void getEs(double *E,
           int nmax, double Elower, double Veff[],
           double r[], double dr[], int N)
```

Output:

- E[n]: linear *array* of energies of states with n=0 nodes through n=nmax nodes (inclusive!)

Input:

- nmax: maximum number of nodes sought
- Elower: energy guaranteed to be lower than any valid solutions
- Veff[]: effective potential on grid
- r[], dr[], N: standard grid information

Write a routine of the above prototype to return the energies of the electronic states with zero through nmax nodes in the potential Veff[]. Your routine should use "rtbisp480" in conjunction with "func_SchrodingerNodes" to first locate appropriate bounding energies (as discussed in lecture), and then use zriddrp480 to find the solution to high precision.

A workable code fragment (but not the most efficient) is

```
/* Loop to get states */
for (n=0; n<=nmax; n++) {
  /* Get E1 as an energy with n nodes */
  E1=rtbisp480(func_SchrodingerNodes,Elower,0.,1e-12,n,Veff,r,dr,N);
```

```
    /* Get E2 as an energy with n+1 nodes */
    E2=rtbisp480(func_SchrodingerNodes,Elower,0.,1e-12,n+1,Veff,r,dr,N);

    /* Now, get the solution, which is in between! */
    E[n]=zriddrp480(func_Schrodinger,E1,E2,1e-12,0,Veff,r,dr,N);
  }
```

Note: All solutions will have E<0, so using [Elower,0.] as the bounds for the "rtbisp480" searches is guaranteed to work!

2.2 Fixing "zriddrp480"

The numerical recipes routine "zriddr" also has a small bug: For wildly varying functions such as "func_Schrodinger", zriddr sometimes returns too early. To fix this, replace the code fragment

```
if (fabs(xnew-ans) <= xacc) ... ,
```

in "zriddrp480" with

```
if (fabs(xl-xh) <= xacc) ... .
```

2.3 Debugging

Write a main program which calls "getEs" to get the energies of the states for Veff[k]=-1./r[k] with 0, 1 and 2 nodes. As usual, your main program should accept the number of intervals N as input from the keyboard so that we may explore convergence.

3 "getallEs"

3.1 New routine

```
void getallEs(double **E,
              int lmax, int nmax[],
              double Z, double V[],
              double r[], double dr[], int N)
```

Output:

- E[l][n]: *matrix* of energies of states of angular momentum l and n=0 nodes through n=nmax[l] nodes (inclusive!)

Input:

- lmax: maximum angular momentum to consider (l=0 through l=lmax)
- nmax[l]: maximum number of nodes sought for angular momentum l
- Z: charge on the nucleus
- V[]: "self-consistent potential (V[k]=-Z/r[k], for now ...)
- r[], dr[], N: standard grid information

Write a routine of the above prototype which returns the energies of the electrons in an atom with potential V[]. The routine should return the results in the form of a double precision two dimensional matrix of values E[l][n]. It should determine the following table of values:

l=0	E[0][0]	E[0][1]	...	E[0][nmax[0]]
l=1	E[1][0]	E[1][1]	...	E[1][nmax[1]]
⋮			⋮	
l=lmax	E[lmax][0]	E[lmax][1]	...	E[0][nmax[lmax]]

As discussed in lecture, your routine should do this by calling "getEs" with

$$V_{eff}(r) \equiv V(r) + \frac{\ell(\ell+1)}{2r^2}$$

A suitable, but not the most efficient, code fragment is

```
double *Veff;
                        .
                        .
Veff=dvector(0,N); /* Must declare space for Veff */
                        .
                        .
  for (k=0; k<=N; k++) { /* Compute Veff */
    Veff[k]=V[k]+l*(l+1)/(2*r[k]*r[k]);
  }
  Veff[0]=0.; /* Mathematically proper for origin */
                        .
                        .
  getEs(E[l],nmax[l],-Z*Z,Veff,r,dr,N); /* Solve for states of ang mom l */
                        .
                        .
free_dvector(Veff,0,N); /* Must free Veff at end of routine! */
```

where, as discussed in lecture, we use -Z*Z as a guaranteed lower bound. Note the convenient use of the *Numerical Recipes* dvector() routine to declare space for Veff.

Hints on working with two-dimensional matrices:

Note that, if you are working in **C**, the above code fragment will only work if E is properly passed in as an array of pointers, which can get quite tricky unless you have a lot of experience with dealing with pointers in **C**. All of this is done for you by the numerical recipes "dmatrix" function, which you should use to form the object E in the routine which calls getallEs. Your main program for (3.2) below, for instance, would look like

```
{
  double **E; /* In the declarations section */
                    .
                    .
  E=dmatrix(0,lmax,0,nmaxmax); /* Among the first executables -
                    .             lmax is the maximum l value we expect;
                    .             nmaxmax is the maximum expected of all nmax[l]'s */
    getallEs(E,lmax,nmax,Z,V,r,dr,N); /* Example call */
    printf("E[l=0][n=0]=%f, E[l=1][n=0]=%f\n",E[0][0],E[1][0]); /* Use E[][] like this */
                    .
                    .
  free_dmatrix(E,0,lmax,0,nmaxmax); /* At end of routine or main, must call free
                                       for each dvector and dmatrix before exiting! */
}
```

If you are working in **Fortran**, then you must reverse the order of the array, so that it is used as E(n,l). The proper declaration of E, call for "getallEs" and write statement for the above example is then

```
double precision E(0:nmaxmax,0:lmax)
            .
            .
call getallEs(E(0,1),lmax,nmax,Z,V,r,dr,N)
write(6,*) 'E(n=0,l=0)=',E(0,0),', E(n=0,l=1)=',E(0,1)
```

Note that, here, the first argument to E is the number "one" and the second argument is the letter "ell". Finally, in **Fortran**, getallEs needs to know the dimensions of E, so you'll have to add one more argument to the prototype as in

```
subroutine getallEs(E,lmax,nmax,Z,V,r,dr,N,nmaxmax)
double precision E(0:nmaxmax,0:lmax)
integer lmax
integer nmax(0:lmax)
double precision Z
double precision V(0:N)
double precision r(0:N),dr(0:N)
int N
```

3.2 Debugging

Write a main program which calls "getallEs" to get the energies of the states for carbon. Carbon has electrons in three states: "1s" (l=0 with zero nodes), "2s" (l=0 with one node), and "2p" (l=1 with zero nodes). For now, take the potential for carbon as V[k]=-Z/r[k], with Z=6. Finally, as usual, your main program should accept the number of intervals N as input from the keyboard so that we may explore convergence.